CHILL OUT: A STUDY ON THE RELATIONSHIP BETWEEN RELAXATION AND ACCURACY LOSS AND THE IMPLICATION THEREOF

An Undergraduate Research Scholars Thesis

by

RYAN GARMESON

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Jennifer Welch

May 2019

Major: Computer Science

TABLE OF CONTENTS

Page

ABSTRACT

The Effect of Relaxation on Correctness of a Single-Source Shortest Path Algorithm

Ryan Garmeson
Department of Computer Science
Texas A&M University

Research Advisor: Dr. Jennifer Welch
Department of Computer Science and Engineering
Texas A&M University

Linearization is the main paradigm by which we implement distributed objects, though it incurs a high synchronization cost. Relaxation can mitigate the synchronization cost of a distributed object. Currently, we lack methods for determining which applications would most benefit from relaxation.

We investigate a specific case of relaxation to clarity to this problem. We examine a relaxed priority queue used in Dijkstra's solution to the single-source shortest path problem. We focus on percent correctness of the resulting paths as a measure of accuracy, examining the trend of accuracy loss with a standard benchmarking problem. We find that accuracy loss is not directly correlated with level of relaxation, implying further study is warranted in order to quantify the relationship between relaxation and accuracy loss so as to be more useful in application.

DEDICATION

To Roxanne, who encouraged me to find passion in my work.

ACKNOWLEDGMENTS

This work would not have been possible without the support of my mentor and advisor, Dr. Jennifer Welch. She has graciously guided me when I have gotten discouraged or stuck or simply gone the wrong way. The critiques and guidance received from her team have also been instrumental in the development of this work as an academic piece.

Additionally, I would like to thank Roxanne, again, for her endless patience and superb editing skills.

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Distributed objects are vital in modern computing, facilitating the development of high-level applications. The common model for implementing distributed objects is linearizability, offering the illusion of a local, non-distributed object; however, linearization has long been known incur expensive synchronization costs, and there is increasing interest in finding ways to relax the notion of linearizability to reduce that cost.

One promising avenue of this research is in relaxation, which is a model for loosening the requirements of linearization to achieve cheaper use while mitigating functionality loss. However, much of the current scholarship on this subject is focused on narrow applications and/or speed improvements. Few, if any, paradigms to calculate the functionality cost of relaxing data structures exist.

In this paper I will implement and test a model of relaxed data structures using a relaxed priority queue; I will discuss the technical implications of my methods, tests, and results; I will examine the effects that the particular graph topologies may have on my results; and, finally, I will present the rules I have found through these experiments and their implications to current applications.

1.1 Background

1.1.1 Distributed Objects

This paper will discuss strategies used in distributed computing; we will examine these strategies in a non-distributed environment and simulate the properties of the distributed environment. Further work should verify that our findings hold true in a distributed environment as well.

1.1.2 Linearization

Linearization is the practice of simulating, to the program, that the program is running sequentially on one processor [1]. Linearization has a high scaling cost, as shown by Alistarh et. al, and the synchronization cost overwhelmingly weighs down the program speed after 10 threads [2]. This is the motivation behind finding some way to avoid entirely linearizing our distributed algorithms.

1.1.3 Relaxation

The strategy of avoiding total linearization that we have chosen to examine is relaxation. To relax an object is to lower the consistency conditions of the object such that the relaxed object operates within some bound of the linearized object [3]. Alistarh and Wimmer [2, 4] both describe the k-relaxation of a queue, wherein the pop function is relaxed such that it may return any of the first k elements of the linearized queue.

1.2 Related Work

Several algorithms have been created to implement a relaxed priority queue. Wimmer created the lock-free k-relaxed priority queue[4]. Alistarh developed Spraylist, which uses a skiplist to implement a similar distributed queue[2]. This paper is not focused on algorithms, and is using a naive algorithm to achieve a similar effect.

This work will not focus on performance improvement of the relaxed algorithms, as that has been proven and tested [2, 4, 3, 5]. It has been found that relaxed structures have a significant performance and scalability increase compared to their linearized relatives.

Current scholarship focuses on speed improvement of relaxation without critically examining the tradeoff of accuracy [2, 4, 3]. This paper will focus on extending knowledge to the realm of practical application, finding the classes of applications which can support relaxing otherwise linear objects without impacting performance and quantifying the

aforementioned tradeoff.

2. METHODS

We began this project with three questions:

1. What applications can effectively use relaxation to cut synchronization time?
2. How do relaxed data structures affect the correctness of the applications that use them?
3. Can we quantify the tradeoff between speed increase and correctness loss?

We designed an experiment to attempt to answer these questions. We wanted to use graphs as a stand-in for different types of applications, to solve a problem with them using relaxed data structures, and to graph the results in order to prove that the tradeoff between speed increase and correctness loss is not linear.

We designed a program to use Dijkstra's algorithm [6] on unweighted graphs to solve the Single Source Shortest Path (SSSP) problem. We created small random graphs and ran them from all possible sources, with all valid k, where $1 \leq k \leq N$. We summed the path lengths from these results along k, so as to compare whether and how $k = 1$, the base case, differed from cases with $k > 1$.

2.1 Implementation

For our implementation, we chose to use a readily-available template of Dijkstra's algorithm from Rosetta Code. We chose this option in order to speed up development time and use moderated, successful code rather than implementing our own. We used the C++ implementation of this algorithm.

Our implementation used a priority queue which contained the frontier of the search only. We believed that this type of implementation, as opposed to one in which all vertices

are initially known to the priority queue, had the greatest likelihood of returning non-linear changes in accuracy. We introduced a relaxation variable, k, which would change the element returned when the priority queue called for min(). The change we introduced was a uniform random selection of the first k elements of the priority queue, modulo the the current size of the queue in order to ensure that something is returned by min(). This is similar behavior to Wimmer's [4] implementation of the distributed, lock free queue.

We used graphs from Stanford's SNAP library as experimental data for our algorithm. We intended to use graphs from different applications in order to directly see how relaxation affected real-world data. We also used several randomly generated graphs for smaller datasets for testing purposes. The testing graphs had an average node degree of 3, 6 nodes each, and 18 edges. Graph drawn in Figure 2.1 and specified in Tables 2.1-2.6.

Due to time constraints, we were unable to bring the full algorithm up to the standards required by our experiment design. We chose to use our testing data in substitution for the larger, more complex graphs in order to meet the time constraints without sacrificing the spirit of the experiment.

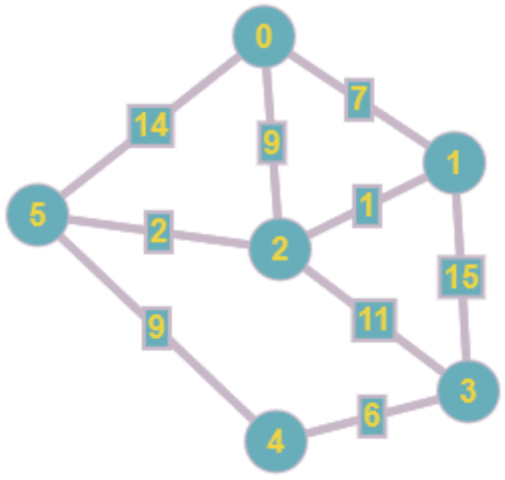

Figure 2.1: The graph used for tests and final run. Created at http://graphonline.ru/en/

Table 2.1: Graph 0. Other graphs are a reshuffle of this graph.

Source	Target	Weight
0	1	7
0	2	9
0	5	14
1	0	7
1	2	1
1	3	15
2	0	9
2	1	1
2	3	11
2	5	2
3	1	15
3	2	11
3	4	6
4	3	6
4	5	9
5	0	14
5	2	2
5	4	9

Table 2.2: Graph 1.

Source	Target	Weight
0	5	7
0	1	1
0	2	15
1	5	9
1	0	1
1	2	11
1	4	2
2	0	15
2	1	11
2	3	6
3	2	6
3	4	9
4	5	14
4	1	2
4	3	9
5	0	7
5	1	9
5	4	14

Table 2.3: Graph 2.

Source	Target	Weight
0	4	9
0	5	1
0	1	11
0	3	2
1	5	15
1	0	11
1	2	6
2	1	6
2	3	9
3	4	14
3	0	2
3	2	9
4	5	7
4	0	9
4	3	14
5	4	7
5	0	1
5	1	15

Table 2.4: Graph 3.

Source	Target	Weight
0	4	15
0	5	11
0	1	6
1	0	6
1	2	9
2	3	14
2	5	2
2	1	9
3	4	7
3	5	9
3	2	14
4	3	7
4	5	1
4	0	15
5	3	9
5	4	1
5	0	11
5	2	2

Table 2.5: Graph 4.

Source	Target	Weight
0	5	6
0	1	9
1	2	14
1	4	2
1	0	9
2	3	7
2	4	9
2	1	14
3	2	7
3	4	1
3	5	15
4	2	9
4	3	1
4	5	11
4	1	2
5	3	15
5	4	11
5	0	6

Table 2.6: Graph 5.

Source	Target	Weight
0	1	14
0	3	2
0	5	9
1	2	7
1	3	9
1	0	14
2	1	7
2	3	1
2	4	15
3	1	9
3	2	1
3	4	11
3	0	2
4	2	15
4	3	11
4	5	6
5	4	6
5	0	9

3. RESULTS

We ran six iterations of our small test using all values of k. We found that, for all values of k, the sums of path lengths returned were identical. This indicates that the relationship between relaxation and accuracy loss is not so simple as being linear with relaxation, as previously assumed. See Table 3.1 for tabular results.

Table 3.1: The sums returned by the program at various k for different graphs.

k	Graph 0	Graph 1	Graph 2	Graph 3	Graph 4	Graph 5
1	6	6	6	6	6	6
2	5	5	5	5	5	5
3	3	3	3	3	3	3
4	5	5	5	5	5	5
5	4	4	4	4	4	4
6	5	5	5	5	5	5

4. SUMMARY AND CONCLUSIONS

Relaxation is a method of mitigating the synchronization cost of linearization, producing a large speedup at the cost of accuracy. Previously, this accuracy loss was not well examined and was assumed to be proportionate to the relaxation of the data structure.

We examined this connection, taking the assumption that problem structure had effect on accuracy loss from relaxation. We used Dijkstra's algorithm to solve the SSSP problem with k, our stand-in variable for relaxation, from 1 to N. We found that for the algorithm we used, no difference was found between the $k = 1$ and $k = N$ runs, implying that accuracy loss is not proportionate to relaxation of the structure.

4.1 Further Study

Our first goal of further study would be in running this experiment as originally designed in order to generate enough data points to quantify the difference between assumed inaccuracy and actual inaccuracy, which would satisfy our third research question and go further towards answering 1) what applications can effectively use relaxation to cut synchronization cost and 2) how relaxed data structures affect the correctness of the applications that use them.

A version of this experiment in which one which uses the full vertex set, rather than the frontier only, could also be worth further study. We suspect that the vertex experiment will be more susceptible to error introduced from relaxation than the frontier version.

REFERENCES

[1] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Transactions on Programming Languages and Systems*, vol. 12, pp. 463–492, jul 1990.

[2] D. Alistarh, J. Kopinsky, J. Li, and N. Shavit, "The spraylist: A scalable relaxed priority queue," *ACM SIGPLAN NOTICES*, vol. 50, no. 8, pp. 11–20, 2015. ID: 000367254800002.

[3] Y. Afek, G. Korland, and E. Yanovsky, "Quasi-linearizability: Relaxed consistency for improved concurrency," in *Principles of Distributed Systems* (C. Lu, T. Masuzawa, and M. Mosbah, eds.), (Berlin, Heidelberg), pp. 395–410, Springer Berlin Heidelberg, 2010. ID: 10.1007/978-3-642-17653-1_29.

[4] M. Wimmer, J. Gruber, J. L. Traeff, and P. Tsigas, "The lock-free k-lsm relaxed priority queue," *ACM SIGPLAN NOTICES*, vol. 50, no. 8, pp. 277–278, 2015. ID: 000367254800038.

[5] E. Talmage and J. L. Welch, "Improving average performance by relaxing distributed data structures," in *Distributed Computing* (F. Kuhn, ed.), (Berlin, Heidelberg), pp. 421–438, Springer Berlin Heidelberg, 2014. ID: 10.1007/978-3-662-45174-8_29.

[6] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. ID: Dijkstra1959.

[7] T. A. Henzinger, C. M. Kirsch, H. Payer, A. Sezgin, and A. Sokolova, "Quantitative relaxation of concurrent data structures," in *Proceedings of the 40th Annual ACM*

SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, (New York, NY, USA), pp. 317–328, ACM, 2013.